Scheduling of Periodic Tasks with Data Dependency on Multiprocessors

Jinlin Wang
Laboratory of Embedded Systems
School of Computer Science and Engineering, Beihang University
100191, Beijing, China
wangjinlin@les.buaa.edu.cn

***Abstract*—This article studies the scheduling problem of a set of tasks with time or data constraints on a number of identical processors with full connections. We present an algorithm, in which a set of static schedule lists can be obtained, each for a processor, such that each task starts executing after its release time and completes its computation before its deadline, and all the precedence relations between tasks resulting from data dependency are satisfied. The data dependency relations between tasks are represented by Synchronous Dataflow Graphs (SDF) as they can indicate tasks' concurrency and enable effective scheduling on multiprocessor platforms. The SDF, however, does not support the time constraints of tasks directly, thus an adaption is applied to conform to the time limits. With this adaption, the periodic tasks of implicit-deadline or constrained-deadline can be scheduled on multiprocessor platform effectively.**

Keywords-multiprocessor; scheduling; real-time; SDF; data-dependency

I. INTRODUCTION

Currently, the studies of multiprocessor scheduling algorithm of real-time tasks are mostly focused on time constraints and the utilization of the processor. The precedence relationship, which arises from inter-task communication between periodic tasks, has not been well discussed yet and a mature solution is still lacking. In the existing reliable embedded real-time operating systems, such as OSEK [1] for automotive, RTEMS [1] for aerospace and VxWorks 653[3] which respecting the ARINC 653 standard [4] for aeronautics, scheduling policies for dependent tasks are not provided directly, meaning that the determinism of task communications is usually ensured manually by the programmers[5][6].

In DSP systems, the existing algorithm is able to better schedule tasks with data dependence relations. Edward Ashford and David G. Messerschmitt in [7] proposed an algorithm to convert an SDF into Directed Acyclic Graph (DAG), and then can get the static schedule lists. However, these scheduling algorithms in DSP are data driven, and they are difficult to deal with real-time tasks which are characterized by time constraints.

There are several related works on this topic before. In order to guarantee the data dependencies between the periodic tasks, in [5], the task time constraints are restricted more strictly: the first release time of the task is put off, and the deadline is shifted to an earlier time. So the weakness of this algorithm is that when there are many data precedencies between tasks, it is likely that they become unable to be scheduled because of the excessively tight time limits. Jia Xu in [6] proposed an algorithm that can statically schedule tasks with release times, deadlines, precedence and exclusion relations, but he did not take into consideration the periodic tasks, which is one of our main objectives. A fast heuristic for parallel software with respect to energy as well as time constraints was presented in [8], in which the constraint specifies how much time can pass between the moment when all data necessary for the execution of the node is available and the moment when the execution of the node finishes. This is also different from our problem.

In our algorithm, the tasks to be scheduled can either have a periodic time constraint or some data dependencies with other tasks. There should be at least one constraint on each of the tasks, either time or data. The resulting schedules are static schedule lists, each for one processor. The advantage of static schedule is that it is easy to be verified whether all the time and data constraints are met and whether the schedule is free from deadlock. This algorithm can be flexible as well, for the tasks can be scheduled preemptive or non-preemptive according to the 3rd step of the algorithm.

In Section II, we describe the problem formally. Then, a brief introduction to the algorithm is presented. In Section III and IV, we discuss the algorithm in detail. An example is given in Section V and the paper is concluded in Section VI.

II. PROBLEM DESCRIPTION AND NOTATIONS

The problem can be described as follows:

- Given a set of tasks $T = \{t_i | i \in [1, n]\}$, each task has a worst case execution time (WCET), which denoted as C_i for task t_i.
- There are two possible types of constraints for each task: time and data dependency. For the time constraint, the task has a set of real-time attributes (T_i, O_i, D_i), that is to say the task t_i is periodic, and the period is T_i. O_i is the release time of the first instance of the task. D_i is the relative deadline of the task. The data dependency constraint is designated in SDF, which will be discussed in next section. There should be at least one constraint on each task.
- The number of processors on the target platform is M. All the M processors are completely connected.

In view of the above conditions, a static schedule $S = \{s_i | i \in [1, M]\}$ should be generated according to the constraints. Schedule list s_i indicates when and which task should be executed on processor i.

Here are some more symbols that to be used later:

- R_i[n] is the release time of the n-th instance of task i.
- F_i[n] is the finish time of the n-th instance of task i.
- D_i[n] is the absolute deadline of the n-th instance of task i.

Our algorithm can be divided into 3 steps: first an SDF of the tasks is constructed with respect to the time and data dependency, then we convert the SDF into a DAG with the method from [7], and finally the schedule is obtained from the DAG using a modified Hu-level algorithm [9]. In the next section, we give a brief introduction to SDF first, and then discuss how to add time constraints in the SDF. A proof of correctness will also be provided.

III. SDF WITH TIME CONSTRAINTS

A. Synchronous Dataflow Graph

The Synchronous Dataflow (SDF) [7] model of computation is widely used for Digital Signal Processing (DSP) applications. In this model, the exchange of data between different parts of a program is clearly revealed. The model consists of nodes, representing different tasks, and arcs between the nodes. Data are exchanged via these arcs in a FIFO manner. Figure 1(a) shows an SDF graph where the data production and consumption between tasks are given as arc annotations at the start-point and end-point respectively. The mark "2D" on the arc, indicating 2 Delays, means that at the initial time of the program, there are only 2 valid data for task B, so the first instance of B can only begin to execute after the finish time of execution of the first instance of task A. This is the precedence relation resulting from data dependency.

B. Time constraint

Generally, if two tasks are connected from A to B by an arc, on which the data production of task A is p, the data consumption of task B is c and the delay is b, then the number of instances of task A on which the j-th instance of task B is dependent can be calculated by the following equation as stated in [7]:

$$d_{ba}[j] = \left\lceil \frac{jc-b}{p} \right\rceil \quad (1)$$

The notation $\lceil\ \rceil$ indicates the ceiling function.

The data transferred through arcs in an SDF can be virtual tokens, in which case we can add new precedence relations between tasks by (1). For example, if task B should be run after every three executions of task A, then a new arc from A to B can be added to the SDF, with the production and consumption labeled as in Figure 1(b). Figure 2 shows a variety of precedence relations that can be noted by SDF.

Based on this idea, in Figure 3, the time constraints (T_i, O_i, D_i) of periodic task t_i can be appended to our SDF as follows:

- Add a new self-dependent virtual task (t_0) node marked as V in the SDF. The WCET, period, offset and deadline of this virtual node are 1, 1, 0 and 1 separately.

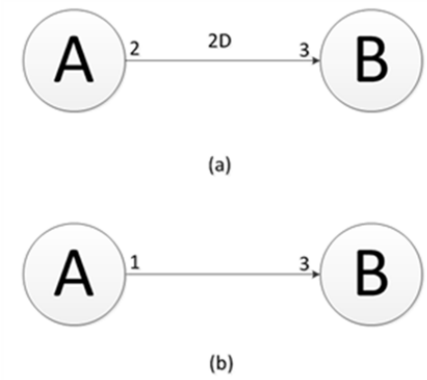

Figure 1. Example of SDF Graphs: (a) Data exchanged via the arc. (b) The first instance of task B can only be executed after 3 executions of task A have finished.

- For each periodic task t_i, add an arc from the virtual node to the task node, on which the data produced by

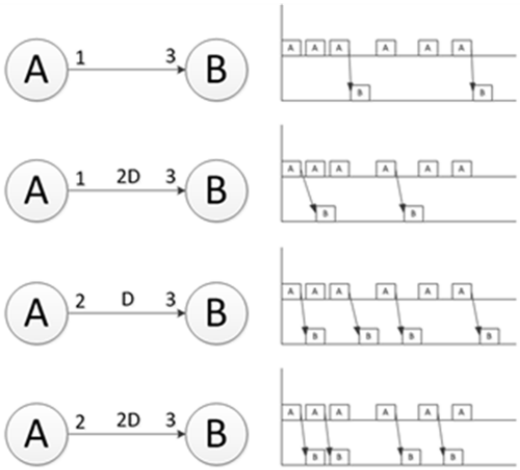

Figure 2. Several precedence relations denoted by SDF.

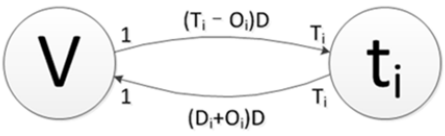

Figure 3. Add time constraints to a periodic task t_i

the virtual node is 1 and the data consumed by node t_i is T_i. The delay on this arc is assigned to

$$d_{it} = T_i - O_i.$$

- For each periodic task t_i, add an arc from the task node to the virtual node, on which the data produced by the task node is T_i and the data consumed by the virtual node is 1. The delay on this arc is assigned to

$$d_{if} = D_i + O_i.$$

- When arranged to scheduling lists, the virtual task instances should be run one after another sequentially without time interval between two adjacent instances, thus the j-th execution of the virtual task is from time (j-1) to time j.

From the last point above, we know that the release time of the j-th instance of the virtual node is

$$R_0[\mathrm{j}] = \mathrm{j} - 1, \tag{2}$$

and the finish time of it is

$$F_0[\mathrm{j}] = R_0[\mathrm{j}] + 1 = \mathrm{j}. \tag{3}$$

C. Proof of correctness

1) Release time constraints

According to the definition of time constraints, the release time of the k-th instance of task t_i is

$$R_i[\mathrm{k}] = (k-1)T_i + O_i. \tag{4}$$

With respect to (1), the k-th instance of task t_i is dependent on

$$d_{i0}[\mathrm{k}] = \left\lceil \frac{T_i k - \mathrm{d}_{it}}{1} \right\rceil = (k-1)T_i + O_i \tag{5}$$

instances of the virtual nodes. And in accordance with (3), the finish time of the $d_{i0}[k]$ times instance of the virtual node is

$$F_0\left[d_{i0}[\mathrm{k}]\right] = d_{i0}[k] = (k-1)T_i + O_i. \tag{6}$$

From (4) and (6), it is clear that $R_i[k]$ is equal to $F_0[d_{i0}[\mathrm{k}]]$. That is to say the start time of the k-th instance of task t_i is later than the finish time of the $d_{i0}[k]$ times instance of the virtual node, which is equal to the release time constraint of task t_i. So the release time constraint is satisfied.

2) Deadline constraints

The absolute deadline of the k-th instance of task t_i is

$$D_i[\mathrm{k}] = R_i[k] + D_i. \tag{7}$$

Considering the $D_i[k] + 1$ times instance of the virtual node, by (1) we know that this node is dependent on

$$d_{0i}[D_i[\mathrm{k}] + 1] = \left\lceil \frac{(D_i[k]+1) - d_{if}}{T_i} \right\rceil = k \tag{8}$$

instances of task t_i. So if all the instances of the virtual nodes can be scheduled sequentially without interval, the finish time of the k-th instance of t_i should be no later than the release time of the $D_i[\mathrm{k}] + 1$ times instance of the virtual node, which is $R_0[D_i[\mathrm{k}] + 1] = D_i[k]$, according to (2). So the deadline constraint of task t_i is satisfied.

IV. Modified Hu-level algorithm

Now we have the modified SDF in which the time constraints have been considered. With the method introduced in [7], the corresponding DAG can be obtained, from which we can determine the scheduling lists of all the processors. Paper [7] also cited the Hu-level algorithm from [9] to translate a DAG into the scheduling lists, but in our DAG, there are some virtual nodes, which should be treated specially.

In DAG, the Static B-level (SBL) of a node is the length of the path from this node to the farthest exit node in the DAG. The length of the path is the sum of the WCETs of all the nodes along the path.

Using the Modified Hu-level algorithm as in Algorithm 1, all the virtual nodes are distributed on the virtual processor while the normal task nodes are scheduled on real processors, and these scheduling lists of the real processors are our final results.

Algorithm 1: Modified Hu-level Algorithm

```
Add a virtual processor P0 to the target platform;
Calculate the static B-Level (SBL) of each node in DAG;
Arrange the nodes in a list according to the descendant order of their SBL;
For each node ni in the list:
    If ni is a virtual node then
        assign this node to P0
        If there is an interval between ni and the previous node on P0 then
            Cannot find a feasible schedule.
            Stop the algorithm.
        End if
    Else
        schedule this node on the processor where the node can start at the earliest time.
    End if
End for
```

Besides Hu-level, there are some other algorithms that can be used to convert the DAG into static scheduling lists. Paper [10] gives a comparison of some of these algorithms, which can also be modified to handle the virtual nodes and thus be used in our algorithm similarly.

V. An example

Here are two periodic tasks: A and B. The time attributes of each task are as follows:

$$C_A = 1, T_A = 2, O_A = 0, D_A = 2$$
$$C_B = 2, T_B = 3, O_B = 0, D_B = 3$$

The data dependency between A and B is represented by SDF in Figure 4. With our algorithm described above, the SDF is modified by adding a virtual node and 4 arcs, as shown in Figure 5. Then in Figure 6, we translate the SDF into a DAG using the method described in [7]. The numbers

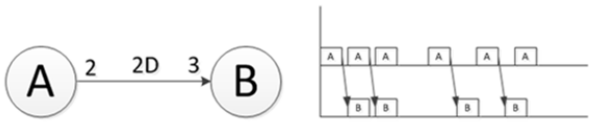

Figure 4. Data dependency between A and B

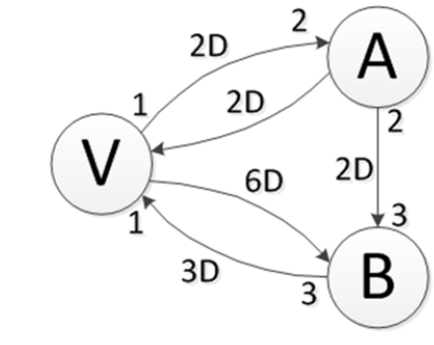

Figure 5. Modified SDF of task A and B

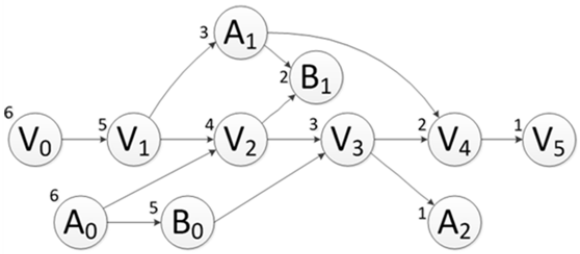

Figure 6. DAG of task A and B.

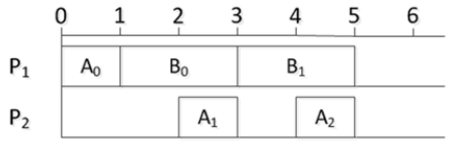

Figure 7. Static scheduling list for each processor.

beside each node of the DAG in Figure 6 is the SBL of it. Finally, the static scheduling lists are determined by the modified Hu-level algorithm, when scheduled on a two-processor platform, as in Figure 7. It is clear that all the task nodes are in their time constraints, and the data dependency is satisfied.

VI. CONCLUSION

In this paper, we proposed a scheduling algorithm for data dependent tasks with real-time constraints. The algorithm is based on SDF. By adding a virtual task node, we made it possible to deal with time constraints as well as data dependency in an SDF graph. To obtain the corresponding static scheduling lists, we also present a modification to the Hu-level algorithm, so that the virtual nodes can be handled specially and the other nodes are scheduled satisfying the constraints.

ACKNOWLEDGMENT

The work is supported by the National High Technology Research and Development Program 863 (No. 2011AA01A204)

REFERENCES

[1] OSEK, “OSEX/VDX Operating System Specification 2.2.1”, OSEK Group, 2003, www.osek-vdx.org.

[2] RTEMS, “RTEMS C user’s guide”, Edition 4.9.2, for RTEMS 4.9.2, OAR Corporation, 2009.

[3] VxWorks, “VxWorks 653 - DO-178B Certified ARINC 653 Real-Time Operating System”, Wind River, 2006.

[4] ARINC, “ARINC Specification 653: Avionics Application Software Standard Interface”, Aeronautical Radio INC, 2005.

[5] Julien Forget, Frederic Boniol, Emmanuel Grolleau, David Lesens and Claire Pagetti, “Scheduling Dependent Periodic Tasks Without Synchronization Mechanisms”, RTAS 2010, pp. 301-310, doi:10.1109/RTAS.2010.26

[6] Jia Xu, “Multiprocessor Scheduling of Processes with Release Times, Deadlines, Precedence, and Exclusion Relations”, IEEE Transactions on software engineering, vol. 19, No. 2, Feb. 1993.

[7] Edward Ashford and David G. Messerschmitt, “Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing”, IEEE Transactions on computers, vol. C-36, No.1, Jan. 1987

[8] Margarete Sackmann, Peter Ebraert and Dirk Janssens, “A Fast Heuristic for Scheduling Parallel Software with Respect to Energy and Timing Constraints”, IEEE International Distributed Processing Symposium, 2011, pp. 1397-1406, doi:10.1109/IPDPS.2011.284

[9] T.C. Hu, “Parallel sequencing and assembly line problems,” Operat. Res., vol. 9, pp. 841-848, 1961.

[10] Ishfaq Ahmad, Yu-Kwong Kwok, “Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors”, ISPAN 1996, pp. 207.